Proceedings of DETC'00
2000 ASME Design Engineering Technical Conferences
September 10-13, 2000, Baltimore, Maryland

DETC2000/DFM-14011

APPLYING CASE-BASED REASONING TO MECHANICAL BEARING DESIGN

Xiaoli Qin[1] **William C. Regli**[2]
Geometric and Intelligent Computing Laboratory
Department of Mathematics and Computer Science
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104

ABSTRACT

Case-Based Reasoning (CBR) provides a promising methodology for solving many complex engineering design problems. CBR is based on the idea that past problem-solving experiences can be reused and learned from in solving new problems. This paper presents an overview of a CBR design system to assist human engineers in performing mechanical bearing design. It retrieves previously designed cases from a case-base and uses adaptation techniques to adapt them to satisfy the current problem requirements. Our approach combines parametric adaptations and constraint satisfaction adaptations. The technique of parametric adaptation considers not only parameter substitution, but also the interrelationships between the problem definition and its solution. The technique of constraint satisfaction adaptation provides a method to globally check the design requirements to assess case adaptability. Currently, our system has been tested in the rolling bearing domain.

INTRODUCTION

Case-Based Reasoning (CBR) techniques are a promising methodology for solving many problems in engineering design. CBR is a subfield of Artificial Intelligence (AI) that is premised on the idea that past problem-solving experiences can be reused and learned from in solving new problems. This paper discusses the use of case-based reasoning techniques to build a CBR system to solve a domain-specific problem — namely, the bearing design problem in engineering applications. This paper presents a three-phase approach to building a practical CBR system for this domain:

1. *Knowledge Representation for Bearing Design Problems*: building a knowledge-base;
2. *Case-Based Reasoning Engine*: design of the reasoner;
3. *Implementation and Examples*: illustrations of how a CBR system can be used during the design phase.

Foundation of Case-Based Reasoning Techniques

The *Case-Based Reasoning Cycle* (1) precisely defines a methodology to build a CBR system for a given domain. A case-based reasoning system can be viewed as a model which is a combination of a *case-base* and *knowledge reasoning* process modules. These modules form a *case-based reasoning shell*, also called a *reasoner*. They are the functions used to manipulate the knowledge in the case-base and they act to *process* user inputs, *recall* similar cases, *retrieve* the most similar case, *evaluate and adapt* the retrieved case and update the case memory. The modules interact with the case-base during processing.

Normally, following problems are involved in a CBR system: **knowledge acquisition, knowledge representation, case retrieval, case adaptation** and **the learning mechanism.**

1. **Knowledge acquisition:** How to acquire useful knowledge

[1]Web: http://www.mcs.drexel.edu/~gxqin; Email: xq22@drexel.edu.
[2]Web: http://www.mcs.drexel.edu/~wregli; Email: regli@drexel.edu.

from application problem domain.

2. **Knowledge representation:** How to use a formal language to represent certain domain knowledge. The knowledge representation theory of case-based reasoning systems primarily concerns how to structure knowledge stored in the case-base to facilitate effective searching, matching, retrieving, adapting and learning. One influential knowledge representation model is the *dynamic memory model* (5). It was developed by Schankand based on his theory, Memory Organization packet (MOP) theory.
3. **Case retrieval:** How to efficiently retrieve from the case-base the case most similar to the current problem. There are two sub-processes involved in case retrieval: one is to retrieve a set of similar cases from case-base, another is to find the most similar case in this set. The first sub-process is accomplished by designing appropriate index scheme for the domain problem. The second task is done using the *Nearest Neighbor Matching Algorithm (NNM)* (4).
4. **Case Adaptation Strategies:** After a CBR system retrieves the most similar case from the case-base, it normally needs to perform adaptation on this retrieved case. There are several adaptation strategies which can be used in a CBR system. They are Simple Substitution, Parameter Adjustment and Constraints Satisfaction (4).
5. **Learning Mechanism:** Learning is the last step in the Case-Based Reasoning system. In a CBR system, after a new problem is solved, the case-base is changed by adding the new case into it. By doing that, the system can retain more and more knowledge along with problem-solving augmentation and achieve learning.

Bearing Design Problems and Knowledge Representation

Bearings are standard mechanical elements that play a very important role in product design and are used extensively in a wide array of mechanical artifacts. They usually support rotating shafts and make relative rotation possible among shafts and other parts, like gears.

Whenever a newly-designed machine requires rotating function, design of its bearings must also be performed. A bearing designed for a certain machine must satisfy the requirements of the overall assembly structure and working environment. The basic way to solve this problem is to perform intensive calculations based on the working conditions and develop a bearing which can satisfy these working requirements. Some computer programs have been developed to help deal with these intensive calculations (3). Although these approaches release human engineers from manual mathematical calculation, they can not perform higher level intellectual actions, like simulating human reasoning process.

This research has taken a different approach to solving the bearing design problem using Case-Based Reasoning (CBR) ... a methodology which provides a very promising way to organize, construct and program the human knowledge into a system. Thus system can contain human experience, and respond to the real world based on its build-in reasoning mechanisms.

Because of complexity of the bearing design problem, the knowledge space in this domain is incomplete and dynamic. Therefore, knowledge acquisition has to be achieved by specifying only the important features of the design problem. Features are only collected if they help solve the specific problem. Other knowledge that is not directly related to solving the problem is discarded. In our approach, we predefine a set of important features for the bearing design problem, and knowledge acquisition is done manually by a knowledge engineer. Because of restrictions mentioned above, the system we built has some limitations. We will briefly talk about them in the last section.

A language called CASL (2) was chosen to represent knowledge pertaining to bearing design in our case-base. The structure of case-base is based on Memory Organization Packet (MOP) theory (5).

Case-Based Reasoning Engine

The case-based reasoning engine is the reasoning system which allows designers to use archived cases to solve bearing design problem. Once domain knowledge has been used to build the case-base, organize memory, build indices, etc., the reasoning engine can execute searches based on the index scheme. The engine can also perform other reasoning processes, including case retrieval, adaptation and system learning.

Our CBR Engine for Bearing Design is implemented with C and the Microsoft Visual C++ programming environment. The kernel of CBR Engine is based on work from the computer science department, University of Wales (2).

KNOWLEDGE REPRESENTATION FOR BEARING DESIGN

Problem Formulation

There are two basic types of bearings commonly used in industry areas: rolling bearings and sliding bearings. Only the former are discussed here. Rolling bearings are further divided into subcategory according to the geometric shape of their rolling components. Some have rolling components that are cylinders and some are spheres, called ball bearings.

The basic components of a bearing are an inner ring, an outer ring, the rolling components and a supporting cage which keeps the rolling components distributed uniformly. In Figure 1, the cages are not shown. Normally, the bearings are installed on a

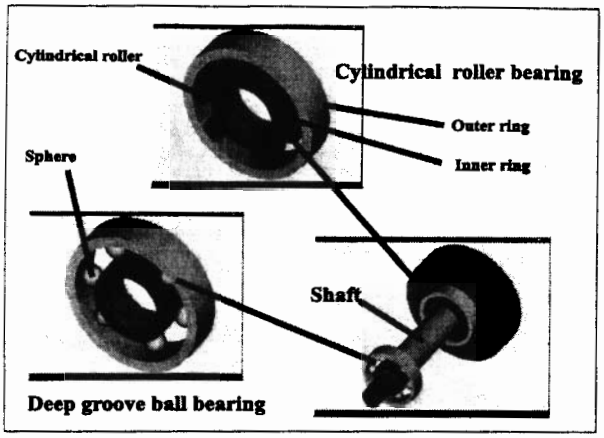

Figure 1. Bearings and where they are installed

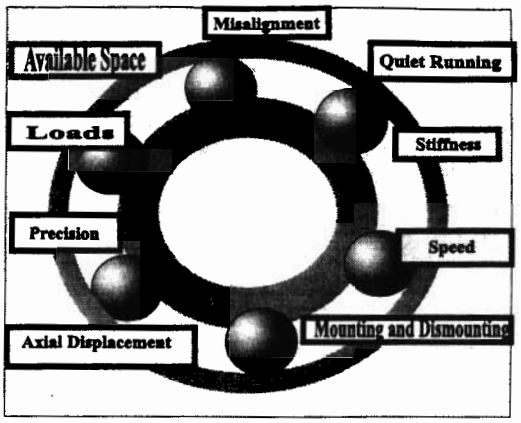

Figure 2. Bearing properties

rotating shaft. The inner ring of a bearing is fastened on a shaft and the outer ring is installed in a housing. The fundamental purpose of a bearing is to transmit the load between a stationary part of a machine (commonly a housing) and the rotating part of the machine (commonly a shaft) with the minimum resistance.

What is meant by a bearing design problem? For purposes of this work, "bearing design" is interpreted from the perspective of an application engineer, i.e., he (she) designs bearings for machines or any applications where bearings are needed. When performing design, he (she) must consider:

1. The working environment for the design problem, including ambient conditions, load conditions, etc..
2. Based on this information and information given by a manufacturer's catalog, which gives different bearings' maximum load capacity, speed limit, etc., how to design and calculate the size of bearing which is suitable for shaft diameter, maximum dynamic life under the working load, maximum speed, etc..

An application engineer's goal in designing bearings for machinery is to make correct decisions in regard to bearing type, size, and material, through analysis of the working environment and extended calculation based on the given working conditions. Appropriate bearing design is vital to the trouble-free operation of machinery.

Important design factors. The inputs of a bearing design problem are the working conditions, including load to be applied on the bearings, shaft speed, lubrication (i.e., oil or grease), assembly space, ambient temperature, corrosive atmospheres and vibrations, etc.. There are also other important factors which must be considered, such as misalignment, quiet running, etc.. Figure 2 shows some primary design factors which a designer might consider.

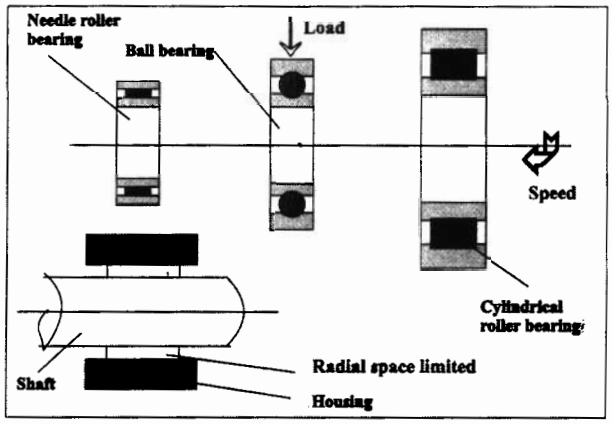

Figure 3. Available space

In this work, we consider the following design factors:

1. **Load,** see Figure 3. The magnitude of the load is the factor which usually determines the size of the bearing to be used. The direction of loads applied on the bearings is also very important.
2. **Speed,** see Figure 3. The speed at which rolling bearings can operate is limited by the permissible operating temperature.
3. **Available Space,** see Figure 3. When radial space is limited, bearings with a small cross section, particularly those with low cross section height, must be chosen. For example, the needle roller bearings are the best choice for this situation.

Calculations. When a designer designs bearings for his machine, not only does he need to consider the factors mentioned above, but he also needs to perform a series of calculations. One of the primary calculations is to predict the probability of bear-

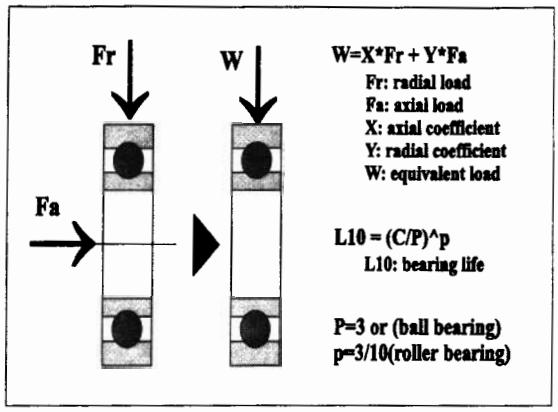

Figure 4. Calculations

ing failure: how long can a bearing be used in a certain working environment? The importance of this calculation is that it is the vital reference for predicting normal operation of the machine, trouble-shooting and maintenance to the machine.

The first step in predicting bearing life expectancy is to calculate the equivalent load applied on the bearing. The Figure 4 illustrates this calculation. Any load applied on a bearing can be decomposed into a *radial load* and an *axial load*. The radial load and axial load are the component forces of a equivalent compound force whose directions are *radial* and *axial*. Normally, a radial load and an axial load can be obtained from a special testing instrument, and the equivalent compound load can be calculated from these measurements.

The variants of the formula given in Figure 4 can be expressed as follows:

Theoretical formula for computing equivalent load applied on a bearing (6):

$$W = \frac{(1-\sin\alpha)F_r + (\cos\beta)F_t}{(2.5-\sin\alpha)}$$

$$W = F_r, \quad \text{if } F_r > W$$

Heuristic formula for computing equivalent load applied on a bearing (6):

$$W = 0.37F_r + 2.0F_t$$

$$W = F_r, \quad \text{if } F_r > W$$

where

W: Bearing load (Newtons)
F_r: Radial load applied to bearing (Newtons)
F_t: Axial load applied to bearing (Newtons)
α: Operating contact angle (radians)
β: Initial contact angle (radians)

The formula for computing bearing life can be expressed as (6):

$$L_n = \left(\frac{C}{W}\right)^3; \text{ (in millions of revolutions)}$$

OR

$$L_{10} = \frac{C^3 * 10^6}{60 * N * W^3}; \text{ (in hours)}$$

where

L_n: The bearing life in millions of revolutions
L_{10}: The bearing life in hours
C: load rating constant (Newtons)
N: Speed of shaft ($\frac{revolutions}{minute}$)
W: The equivalent load imposed on the bearing (Newtons)

Although there do exist other calculations involved in bearing design problems, these calculations are omitted in order to simplify discussion of the domain knowledge.

The Representation Schema

The knowledge pertaining to bearing design problems can be represented in any kind of representing language. Case-Based Reasoning Language (2)(CASL) is used to represent bearing design domain knowledge. CASL is a language specially used for Case-Based Reasoning. The contents of a case-base are described in a file known as a case file, using the language CASL. The reasoner uses this case file to create a case-base in the computer's memory, which can then be accessed and adapted in order to solve problems using Case-Based Reasoning techniques.

General Syntax and Semantics of CASL . Like any other representing language, CASL has strict syntax, semantics, keywords and operators. The syntax of CASL specifies the grammar rules of organizing knowledge, and the semantics of CASL give the concise interpretation of a sentence written in CASL with correct grammar. CASL defines some basic types in the language: identifiers, strings, numbers and operators, etc..

CASL normally divides a case-base into several modules, each of which has its own syntax features and semantic explanations. These modules are the following:

[Introduction]
Case Definition
Index Definition
[Modification Definition]
[Pre-processing Rule Definition]
[Repair Rule Definition]
Case Instance

CASL semantics define the meaning of a sentence by specifying the interpretation of the keywords and basic types, and specifying the meanings of operators. In the syntax blocks of CASL, all keywords and literals are given in bold type. The brief explanations of primary modules are given below:

1. Introduction. This module defines introductory text which is displayed when the reasoning process (reasoner) is run. The purpose of the text is to help the user understand the contents of the case-base or anything else of note.

2. The Case Definition. The purpose of this block is to define the problem features contained in a case.

3. Index Definition. The purpose of this module is to define which fields are to be used as indices.

4. The Adaptation Rule Definition. The purpose of this block is to define rules used to modify a retrieved case from the case-base to make it fit the current problem specifications. The *global repair rule definition* defined in this module allows adaptation rules to be applied on any modified case. The rules defined here are derived from domain knowledge, formulae and constraints.

5. Case Instance Definition. The purpose of this block is to define the structure of a case instance. A case must contain two parts: the problem part and the solution part. The *local repair rule definition* defined in this module allows adaptation rules to be associated with a case. These rules are invoked after the *global* adaptations have run their course.

Examples for Bearing Design Domain Representation.

The feature definitions for user input specifications. When a bearing for a machine is designed, certain working conditions are specified. These specifications are the input to the problem solver, or *CBR reasoner*. The **"case definition is"** block in CASL is used to structure the input specifications. It structures the knowledge about case instances and input problems by defining the primary features of a problem. Following the keyword **case definition is** is the definition of problem features. They can have different *weights* according to their importance in the problem definition. The keyword **weight is** is used to specify the weight of a feature.

In the bearing design problem, the most important features are axial load and radial load. These features' weight values are set to be 5 (reference weight). Load direction, shaft housing diameter, allowed radial limited space, etc., are not that important, comparatively speaking. Therefore, their weight values are set to be 0 (reference weight). A sample case definition using CASL is given below:

case definition is
field shaft_housing_diameter **type is** (d_12_24,d_12_28) weight is 5;
field load_direction **type is** (radial,axial,combined) weight is 0;
field radial_limited_space_requirement **type is** (Yes,No) weight is 0;
field radial_load **type is number** weight is 5;
end;

Some explainations are given below:

1. The feature *shaft_housing_diameter* defines shaft and housing diameters. The purpose of this field is to define a series of possible shaft and housing diameters which may appear in the problems.
2. The *load_direction* field defines the load direction which is applied to the bearing. The purpose of defining this field is that some bearings can only carry axial direction loads, some can only carry radial direction loads, and some can carry loads in both directions.
3. The feature *radial_limited_space_requirement* defines the available radial space in the machine in which the designed bearing can be assembled. In some cases, the design has certain assembly space requirements for special purposes. That is, the available space for bearing design may be restricted in a certain dimension. These space requirements can help a designer predetermine his choice of bearing.
4. The field *radial_load* defines the magnitude of the load which is applied to the bearing in the radial direction. This is the most important factor in deciding the bearing design for a machine, so we specify the field weight as 5 (reference weight).

The index feature definition. This part defines the fields which are used as indices when searching for a matching case. The index scheme defines the methods by which the reasoner should access the case memory. Indices are intended to streamline the matching process. The index features are parts of the

new problem specification. For example, we use the features *shaft_housing_diameter* and *load_direction* as main indices to search the knowledge-base. The sample representation is given below:

```
index definition is
index on shaft_housing_diameter; index on
load_direction;
```

The definitions of adaptation rules. When the old bearing design whose "description of problem definition" part is the most "similar" to the current problem definition is retrieved from the case-base, its solution part must be modified to fit the current problem definition. The reasoner performs adaptations to an old solution according to certain rules defined by domain experts. The **repair rule definition is** block of CASL can be used to define those rules. In the bearing design problem, the following rules (strategies) are defined:

1. Perform simple parameter substitution: substitute parameters of old problem definition into new user input.
2. Perform old solution adjustment to make it fit substituted user input (current problem) according to domain formulae.
3. Check global constraints defined in the case-base to guarantee that no conflicts result.

In the sample given in Algorithm 1, the *change_value_1* is an adaptation rule. It tests a certain condition (represented by a formula) first; when the condition is satisfied, the action is fired. The action here is the recalculation of bearing life (represented by a formula) according to the current user input.

Algorithm 1: Adaptation knowledge representation:

```
(1)  repair rule definition is
(2)  repair rule change_value_1 is
(3)  when
(4)  (0.37 * radial_load + 2 * axial_load)
     ≥ radial_load
(5)  then
(6)  evaluate bearing_life to
(7)  10^6 * support_value_dynamic_C^3
     ---------------------------------------
     (0.37*radial_load+2*axial_load)^3 60*average_speed
(8)  repair;
(9)  end;
(10) end;
```

The definition of a case stored in the case-base. The past experiences of bearing design for applications are stored in the case-base. Representation of these experiences requires the design of certain structures which can represent cases properly. Normally, an experience (case) includes a problem statement part and a solution part. The **case instance is** block of CASL provides a kind of structure and function. This block defines the same structure of problem statement as the **case definition is** block defines.

In a bearing design for an application, some relationships between the problem statement and the solution are unique only for this design (case). For this reason, some features of a case are defined as "local", meaning the attributes for these features are valid only for this design. For example, the features *average_speed* and *expected_bearing_life* are defined as "local" because every bearing designer specifies his own shaft speed and requires his own expected bearing life. Also, every bearing has its own permissible speed limitation defined by the manufacturer and its own life expectancy according to the working environment.

If it is necessary to define some rules to adapt "local" features, then these rules must be specified as "local". That is, the "local rules" are defined in a **case instance is** block. In the given sample below, the rules *rule_1* is "local" because this rule check constraints for "local" features: *expected_bearing_life* . A sample representation of a case is Algorithm 2:

CASE-BASED REASONING SYSTEM

System Overview

The *case-based reasoning engine*, also called *reasoner*, presented here is a program with a Graphical User Interface. It takes problem specifications and a case-base file as its inputs, performs reasoning about the problem, and returns an answer to the user automatically. The reasoning engine of a case-based system consists of four process modules; each of those modules performs certain functions. The modules interact with the case-base and form a reasoning cycle. The first module, *Retrieved case*, takes the current problem specifications as input and outputs a retrieved case. The second module, *Solved case*, decides whether a retrieved case needs to be adapted. This module either returns to the user a solution without further modification or passes a solution to the next module which will perform adaptation on the case. The third module, *Repaired case*, performs this adaptation and returns an adapted case to the next module. The fourth module, *Learned case*, decides whether this new resolved case needs to be stored in the case-base.

The following sections will present how these modules were implemented in our system.

Algorithm 2: Case Instance Representation:

```
(1)   case instance  needle_roller_hk1512
      is
(2)   shaft_housing_diameter = d_15_21;
(3)   load_direction = combined;
(4)   radial_limited_space_requirement = Yes;
(5)   axial_limited_space_requirement = No;
(6)   radial_load = 550;
(7)   axial_load = 100;
(8)   local field definition is
(9)   field average_speed type is number;
(10)  field  expected_bearing_life
      type is number;
(11)  solution is
(12)  bearing_type = needle_roller_hk1512;
(13)  calculation_speed = 10000;
(14)  drill_hole_diameter = 15;
(15)  outer_diameter = 21;
(16)  width = 12;
(17)  support_value_dynamic_C = 7650;
(18)  permissable_speed = 11000;
(19)  bearing_life = 11350;
(20)  local repair rule definition is
(21)  repair rule rule_1 is
(22)  when
(23)  expected_bearing_life ≥ bearing_life
(24)  then
(25)  pr 'Abandon your selection ! ';
(26)  pr 'Bearing life can not meet your requirement!';
(27)  reselect;
(28)  repair;
(29)  end;
(30)  end;
```

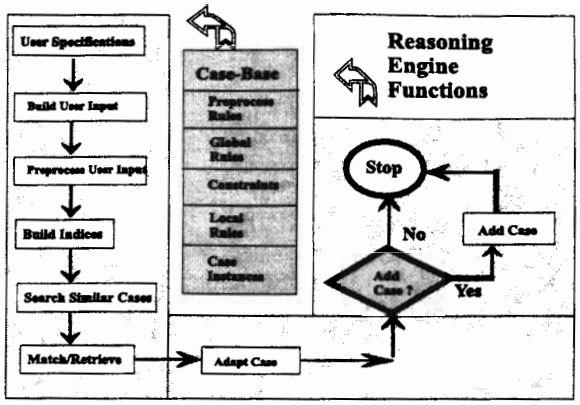

Figure 5. The primary functions of a CBR Reasoning engine

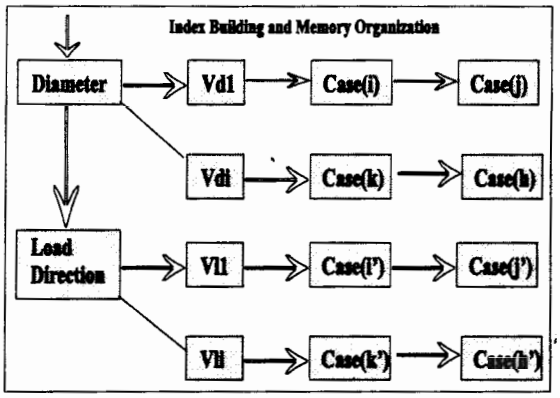

Figure 6. The index building and case-base memory organization

Main Reasoning Engine Algorithm

The flow-chart in Figure 5 shows the main algorithm behind the implementation of a reasoning engine. The two hollow arrows in the figure illustrate that the reasoning engine must interact with the case-base.

The flow-chart shows that the requirements of a module can be broken into pieces or procedures called by the main function. It also shows that a CBR engine forms a reasoning loop. This reasoning loop begins with the procedure *User Specification* and ends with the procedure *Add Case*. Primary procedures used in the main algorithm is discussed below separately.

Building the Index

The performance of a CBR system is determined by the CBR reasoning engine whose efficiency in turn is determined by the design of the *index scheme* and *case-base memory organization*. The index scheme design includes how to specify index features and how to build them in computer memory. The index features are set by domain experts and are represented by the block **index definition is** of CASL. The procedure *Build Indices* takes the representations of index features as input and uses these to build the index scheme. A *linked-list* data structure was chosen to hold the index feature input. The procedure *Build Indices* places all the index features into the linked-list, and at the same time, builds the case-base memory organization. Figure 6 on Page 7 illustrates these ideas.

In this CBR system, for bearing design, two features have been specified as index features: *shaft diameter* and *load direction*. Each index feature is a node of the linked-list; the data type for the nodes is the **struct** type in C. The fields of the **struct** are used to hold attributes of the index features. Figure 6 shows this data structure for the index features and case-base memory. The procedure *Build Index* first links the index features *shaft diame-*

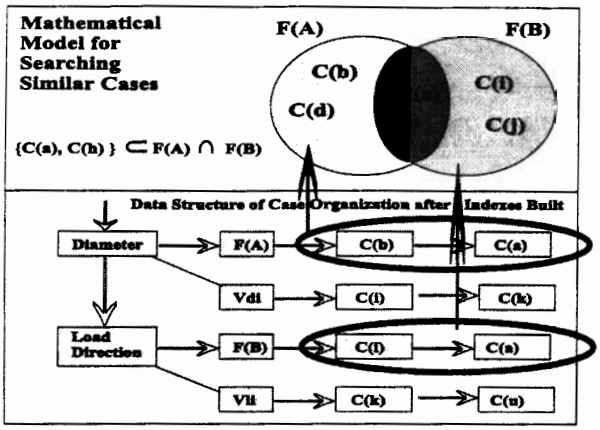

Figure 7. The mathematical model and an example for searching similar cases

ter and *load direction*. It then checks every attribute of the index features. For each attribute, *Build Index* searches for all the cases with the same attribute value in the case-base file and links all of these together.

Case Matching, Ranking and Retrieving

The purpose of building an index scheme is to speed up *searching*. Here, *searching* means to find a set of cases from the case-base which are similar to the current input case. However, the goal here is to find the case which has the maximum similarity to the input case. Thus, a mechanism to rank the similarity of cases is needed. In this section, we discuss how to achieve these two goals: finding a similar case set and finding the most similar case in this set.

First, a mathematical model is presented to show how to find a set of similar cases in the case-base. What are *similar cases*? **Given an input case with certain index features and their attributes, similar cases are those cases whose index features and attributes are exactly the same as the corresponding input case's.** Figure 7 shows these ideas.

The upper part of the Figure 7 presents the mathematical model for finding similar cases. The left and right circles represent attributes $F(A)$ and $F(B)$ of index features A and B of an input case respectively. The $C(n)$ represents a case n. If the left circle includes $C(b), C(d), C(h)$ and $C(a)$, which are the cases with attribute $F(A)$ of feature A, and the right circle includes $C(i), C(j), C(a)$ and $C(h)$, which are the cases with attribute $F(B)$ of feature B, then their intersection contains cases $C(a)$ and $C(h)$, which have both attribute $F(A)$ and $F(B)$. This can be represented in set theory:

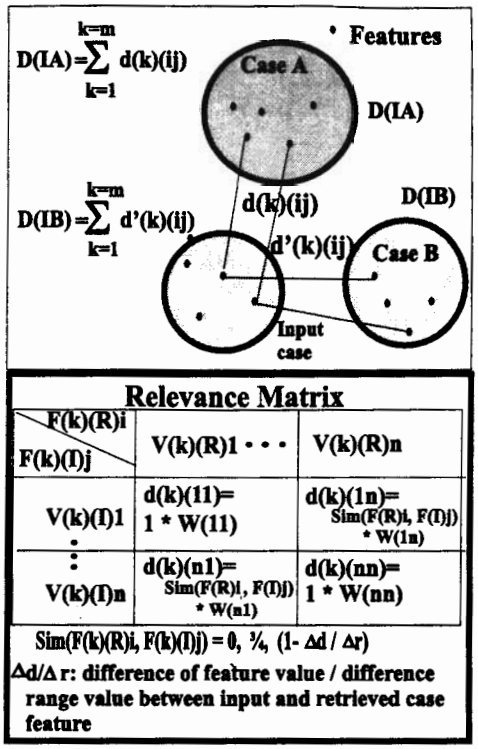

Figure 8. The Nearest Neighbor Matching algorithm

$$\{C(a), C(h)\} \subset F(A) \cap F(B)$$

The lower part of the Figure 7 gives a corresponding example which illustrates how this process occurs in the case-base.

After all similar cases are found, a mechanism to find the most similar case in this set is needed. We used the **Nearest Neighbor Matching algorithm (NNM)** (4). Figure 8 shows how this algorithm works in our CBR system for bearing design. To simplify discussion, we assume that all the component loads (axial load and radial load) applied on the bearing are at the same direction.

The basic idea of the NNM algorithm is to compare the attribute value of each feature of each case in the set of similar cases to every corresponding feature's attribute of the input case, calculate the comparison values and then sum them for each case to get a total comparison value.

In the upper part of Figure 8, the circles represent cases, the dots represent attribute values of features, index i represents the input case, and index j represents cases in the set of similar cases. The index k represents the features in a case. The case A and case B in the figure are the cases from the similar cases' set.

The function $d(k)(ij)$ represents the attribute's comparison value of one of the features (feature k) between the input case and case A, which is equal to the formula (4):

$$W(ij) * Sim(F(k)(R)i, F(k)(I)j)$$

where:

k: a feature of a case.

$W(ij)$: the weight of a feature, defined in the case-base file.

$Sim(F(k)(R)i, F(k)(I)j)$: the degree of similarity between one of the features in the input case and the corresponding feature in a case from the similar case set.

The total attributes' comparison value for a case is D(k)(IA), which is equal to the numeric function

$$\sum_{k=1}^{n} W(ij) * Sim(F(k)(R)i, F(k)(I)j)$$

After finishing all calculations, the NNM algorithm selects the case which has the highest value of $D(k)(ij)$ to be the most similar case.

The key thing in the NNM algorithm is the calculation of an attribute's comparison value for a feature between a similar case and the input case. A matrix called the *relevance matrix*, shown in the lower part of Figure 8, is used to explain how to calculate every feature's attribute comparison value. In the matrix, $F(k)(R)i$ means "the feature k of a case from the similar case set which has possible attribute i, where the range of i can be from 1 to some finite number". $F(k)(I)j$ has a similar meaning except in reference to the input case. So, the first row of the matrix represents all the possible attributes of feature k of a similar case, and the first column represents all the possible attributes of feature k of the input case. The intersection of row and column is the comparison value of the feature k. The $W(ij)$ is the weight of a feature in a similar case. The degree of similarity $Sim(F(k)(R)i, F(k)(I)j)$ has three possible values. First, if two features match exactly, the degree of similarity equals 1. Second, if two *abstract symbols* are similar, its value is $\frac{3}{4}$. Third, if two *numbers* are similar (i.e., both fall within the range defined in the *modification* block), then a value is calculated which reflects how close they are in proportion to the range. Then, the $Sim(F(k)(R)i, F(k)(I)j)$ can be calculated by:

$$1 - \frac{\Delta d}{\Delta r}$$

where: Δd is the difference of the feature values between the input case and the retrieved case and Δr is the difference range value. For example, if the attribute value of feature *radial load* for the input case is 100 Newtons, and the corresponding value for a similar case is 120 Newtons, then Δd = 120-100=20. If the definition for the range of similarity is from 90 to 140, then Δr = 140-90=50. *Similarity between 100(input) and 120(a similar case)* $= 1 - \frac{20}{50} = 0.6$

Algorithm 3: Case matching, ranking and retrieving:
Input: User's input problem specification.
Output: The retrieved case with highest weight.

```
MATCHING_RANKING_RETRIEVING(UserInput)
(1)   begin
(2)   while true
(3)     do
(4)     Index_List_Searching( );
(5)     Case_List_Searching( );
(6)     Computing_Weight_Cases( );
(7)     if Case_Matching_Exact = True;
(8)       return Retrieving_Case();
(9)     else
(10)      Evaluating_Similar_Cases( );
(11)      Retrieving_Heaviest_Case( );
(12)      end
```

The **Algorithm 3** defines the functions which implement the finding of similar cases and the most similar case as mentioned above. The procedure *Index_List_Searching( )* performs searching on the linked-list of index features. Procedure *Case_List_Searching( )* searches out cases whose attribute value for certain features is the same as the input case's. Procedure *Computing_Weight_Cases( )* performs calculation of the weight of a retrieved case and returns this. Procedure *Evaluating_Similar_Cases( )* performs ranking for a case with a weight. Procedure *Retrieving_Heaviest_Case( )* retrieves the case with the highest rank and returns this.

Adaptation of Cases

Very rarely, a retrieved case is exactly the same as the newly defined problem. Most of the time, however, the retrieved case is only a similar situation, and so problem definitions and corresponding solutions need to be modified so that the modified case fully fits the current situation and its solution fully satisfies the current problem requirements. This procedure as a whole is called the case adaptation (repair) process. A series of rules are defined for adapting cases. These rules are provided by domain experts or domain axioms and are applied to each case whenever it is necessary.

Adaptation rules are divided into *global rules* and *local rules* The reasoner uses *global rules* to examine the problem fields and

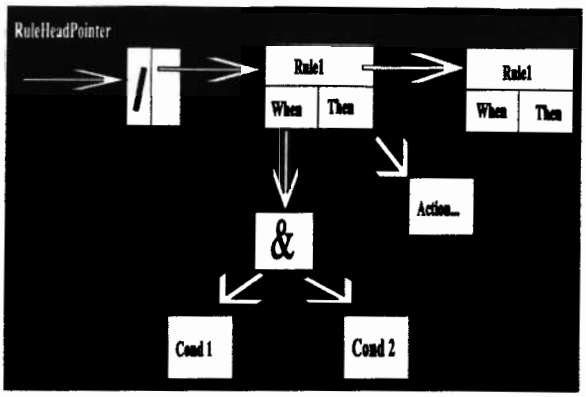

Figure 9. The data structure of global and local rules

solution fields of the retrieved case. These rules are also used to adapt the parameters of the retrieved case and check constraints satisfaction conditions which are specified by the knowledge-base. If there are any constraint conflicts, the repair rules provide a new problem-solving proposal. Otherwise, they adapt the solution of the retrieved case to the new problem. The sample adaptation rules for global repair are described in **Algorithm 1**.

After the reasoner finishes checking the global rules, it immediately checks the local rules defined in the retrieved case. It applies these local rules to the retrieved case to perform local adaptation (i.e., unique to this case). Some sample local adaptation rules are given in **Algorithm 2**.

Figure 9 shows that a linked-list data structure is used to store these adaptation rules. In the figure, every node has two fields: one stores the condition of a rule, the other stores the action. The procedure given in **Algorithm 4** scans the rule list repeatedly as it performs adaptation on a retrieved case; if the condition part is true, it executes the corresponding actions on the case.

IMPLEMENTATION EXAMPLES

In this section, the implementation examples of CBR system is presented. We use screen shots to show how the implementation is carried out.

Case-Base Building

First, the system allows designer to choose search method. This function provides the designer the flexibility to search case-base according to his own needs. If the designer chooses the "Search for matching case", the system will ask designer to input problem definitions. If the designer selects "Search specifying indexes separately", the system will ask designer to specify

Algorithm 4: Algorithm for case adaptation:
Input: Retrieved case.
Output: The modified case.
CASE_ADAPTATION(*RetrievedCase*)

```
(1)   begin
(2)   while true
(3)      do
(4)      if Global_Rules = True;
(5)         Finding_Global_Rule_Headpointer( );
(6)         Searching_Global_Rules( );
(7)         Apply_Modifying_Retrieved_Case( );
(8)         Parametric_Adaptation( );
(9)         Constraints_Adaptation( );
(10)        Evaluating_Solutions( );
(11)     else
(12)        Finding_Local_Rule_Headpointer( );
(13)        Searching_Local_Rules( );
(14)        Apply_Modifying_Retrieved_Case( );
(15)        Parametric_Adaptation( );
(16)        Constraints_Adaptation( );
(17)        Evaluating_Solutions( );
(18)        return Modified_Satisfied_Case;
(19)        end
```

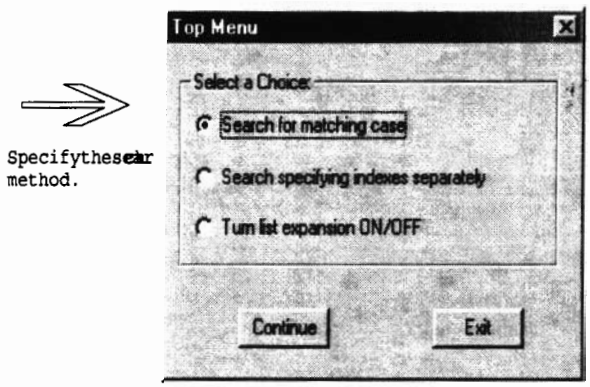

Figure 10. Window for selecting searching methods

the indexes and their values he wants to use. See Figure 10 to illustrate how to select searching methods. In this window example, we select "search for matching case".

Problem Specifications

Global Problem specifications Knowledge acquisition is implemented by system interacting with user. So the following window provides designer to input his problem specifications, like shaft (bearing bore) diameters, load direction, allowed bearing

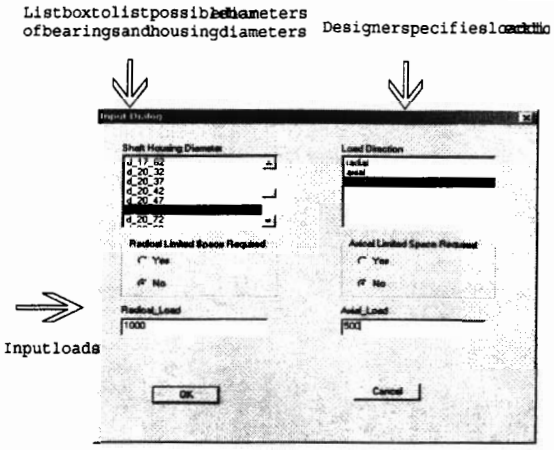

Figure 11. Window for problem specifications

axial/radial space and the amount of loads. Here, different inputs will bring up different other windows and message boxes to indicate different reasoning results. In this window example, we input follow parameters: See Figure 11 to illustrate using example parameters.

Shaft (Bearing Bore) and Shaft Diameter : d-20-52, which means that the shaft diameter (bearing bore diameter) is 20mm and housing diameter is 52mm.

Load Direction : Combined, which means that the loads applied on bearing are combined load (can be decomposed into axial load and radial load).

Required Radial Space: No, which means that bearing is designed without radial space requirement, that is, the bearing is rigidly mounted on shaft.

Required Axial Space: No, see above explaination.

Radial Load: 1000, which means the external radial load applied on bearing is 1000 Newton.

Axial Load: 500, which means the external axial load applied on bearing is 500 Newton.

After designer inputs all above parameter, the system will perform following actions:

1. Searching index linked list which index feature is shaft(bearing bore) diameter and housing diameter, their values are 20mm and 52mm.
2. Searching index linked list which index feature is load direction, its value is "combined".
3. Linking all the cases satisfy above indexes requirements.
4. Comparing other features to each case selected according to index features
5. Calculating the weight of each case.
6. Listing priority of each case.

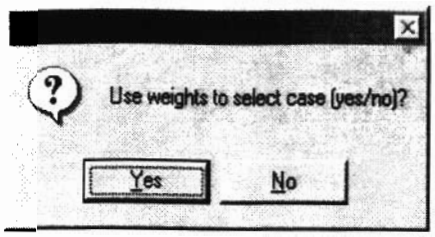

Figure 12. Window for using Weight Algorithm

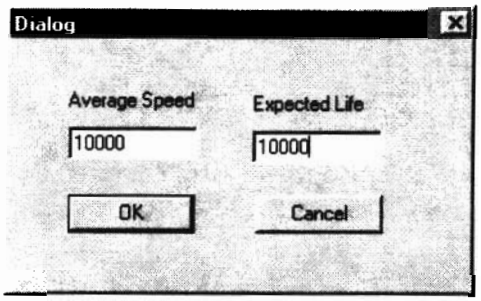

Figure 13. Window for inputting local problem specification

Local Problem Specifications

After the designer finishes his global problem specifications, the another message box will be brought up. It asks whether designer wants to use "Weight Algorithm". If the designer answer is Yes, the system will perform actions based on NNM algorithm (4). If the designer's answer is No, the system will simple assign all the features' weight as 0, and find similar cases based on numbers of matched features. See Figure 12 on Page 11 to illustrate message box which ask user whether he wants to use weight algorithm. In this window example, we select "Yes".

In this system, there are two local fields which define the features being specific for each case. In this window, we input the rotation speed of shaft and the bearing life that the designer is expected. See Figure 13 to input local problem specification.

Adaptation

While all of the details of the adaptation procedures are hidden from the designer, the system presents a series of message boxes which let designer know which case it is using to performing adaptation. In addition, we keep track which cases have failed during adaptation. This loop continues until the system finds a case which satisfies problem specification or announces it failed to find any case that could fit the current problem. See Figure 14 to keep track the adaptation for cases.

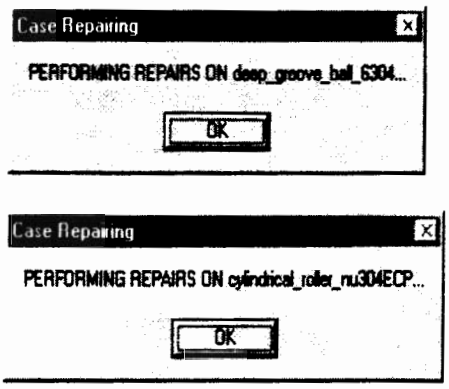

Figure 14. Message box shows the system is performing adaptation on a retrieved case

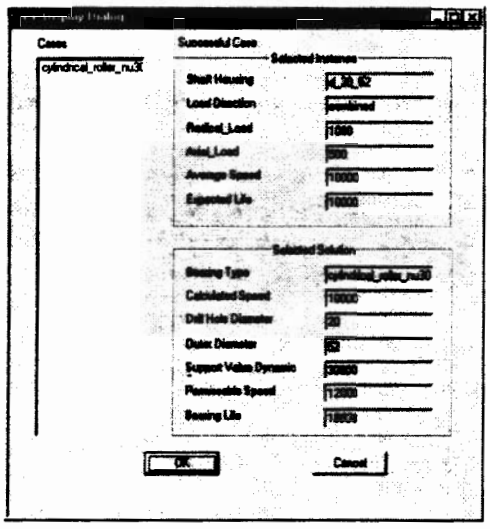

Figure 15. Window shows the successful case

After the system has found a set of retrieved cases and performed successful adaptation on one of these cases, it automatically returns the adapted case. The system can also return a successful or failed case to the designer, allowing the designer to understand why the case is successful or why the case is failed; and so the designer can use this case as a starting point for designing. Figure 15 keeps track the adaptation for successful cases. Figure 16 shows a case that is failed for adaptation.

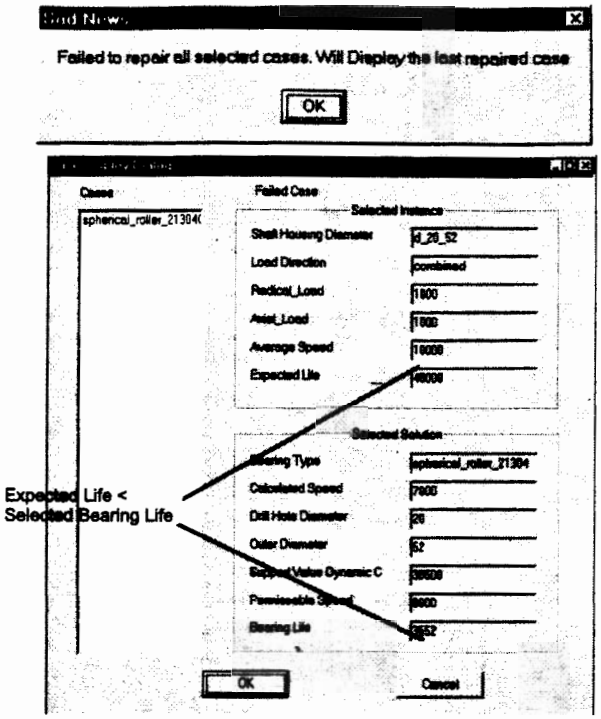

Figure 16. Window shows the failed case

CONCLUSIONS AND FUTURE WORK

Conclusions

This article discussed a system that uses Case-Based Reasoning as both a cognitive model and problem solving methodology to deal with the bearing design problem in mechanical engineering applications. We believe that this work has produced several insights into how AI and CBR techniques can be better applied to more realistic engineering problems:

1. **Knowledge Capture:** Because the knowledge space for the bearing design domain is extremely incomplete and dynamic, it is difficult to formalize general, *a priori*, rules to help the designer solve problems or automate the design process. In contrast, by using CBR techniques, a set of bearing design experiences can be stored in a case library to guide the designer. Through building a knowledge acquisition system, an autonomous CBR intelligent system can evolve and grow more easily than a traditional knowledge-based system.
2. **Adaptability:** CBR techniques can integrate knowledge acquisition, reasoning mechanisms, knowledge storage and learning in one platform. Therefore, a system using CBR techniques can possibly grow and be expanded to encom-

pass a wider variety of assemblies without changing the fundamental system structure.

3. **Augmenting Intelligence:** Our system, rather than being completely autonomous, interacts with the user to obtain knowledge. It provides the flexibility to draw design conclusions either from the reasoning system itself automatically or by allowing the designer to choose a past case as his problem solution directly.
4. **Human-Guided Search:** Our system also provides the flexibility to allow the designer to loosen index constraints to continue reasoning when an exact searching fails. In this manner, the designer has the most opportunities to obtain a design solution which is useful for his current problem. This solution also can be used as a reference for his current design.

Contributions and Future Work

The contributions of this research touch on both AI/CBR and engineering design. We view the system for Case-Based Bearing Design as a template for other CBR environments to create design aides focused for different design problems. We see the following areas for future research.

1. **Knowledge engineering issues:** Because of the limitations of the CASL used to build our system, there are still many limitations in expressing design intent. The case collection process is quite complicated and inefficient, and case-base maintenance is very unstructured, which makes debugging the case-base very difficult. Better methodologies for case collection and good protocols to maintain the case-base are needed.
2. **Knowledge acquisition issues:** We built attribute (features) pairs at design time to allow the user to interactively input his knowledge. If the system is expanded (especially cross-domain), it would be very difficult to enumerate all the features at design time to cover any and all possible problem specifications. Therefore, the development of an autonomous knowledge acquisition system is a future challenge for us.
3. **Indexing issues:** We built a fixed feature-based index scheme at design time to speed up searching. However, as stated above, if the system is expanded, it would be impossible to optimize this choice of index features. Also, at run time, many other features may become important primary design factors, but, since these features are not coded into the case-base, the system will fail to find cases which have these important features. How to develop a dynamic index scheme to facilitate this situation will also be a challenge for us.
4. **Graphical reasoning issues:** Since almost every designer uses CAD or other graphical software to conduct his design, how to combine textual reasoning procedures with graphical reasoning procedures is another very important issue.
5. **Cross-Domain reasoning issues:** The system presented in this article operates in a very specific domain; expansion of this system to other similar design domains is an important area to explore as well. Since we will correspondingly need to develop cross-domain knowledge representations and adaptations, a cross-domain reasoning system becomes very complicated, but also very useful.

Acknowledgements. This work was supported in part by a National Science Foundation (NSF) CAREER Award CISE/IRIS-9733545 and Grant ENG/DMI-9713718; additional support was provided by the National Institute of Standards and Technology (NIST) under Grant 60NANB7D0092 and AT&T Labs' Internet Platforms Division.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the other supporting government and corporate organizations.

REFERENCES

Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7:39–59, 1994.

Center for Intelligent System. University of Wales. http://www.aber.ac.uk/ dcswww/Research/arg/cbrprojects/getting_caspian.html. Caspian.

HEXAGON. http://www.hexagon.de. Bearing calculation, 1999.

Janet Kolodner. *Case-based Reasoning*. Morgan Kaufmann Publishers, Inc., San mateo, CA 94403, 1993.

C.K. Riesbeck and R.S. Schank. *Inside case-based reasoning*. Erlbaum, Northvale, NJ, 1989.

Donald F. Wilcock and E.Richard Booser. *Bearing Design And Application*. McGraw-Hill Book Company, Inc., 1957.